Factory Pattern – creëren van klassen

Voorbeeld "pizzeria"

Stel je runt een pizzeria en als vooruitstrevende eigenaar, schrijf je de volgende code:

```
public Pizza orderPizza(String type)
{
        Pizza pizza;
        switch (type.toLowerCase())
        {
                case "cheese": pizza = new CheesePizza(); break;
                case "greek": pizza = new GreekPizza(); break;
                case "pepperoni": pizza = new PepperoniPizza();
                                        break;
                default: pizza = null;
        }
        if (pizza != null){
                pizza.prepare();
                pizza.bake();
                pizza.cut();
                pizza.box();}
        return pizza;
}
```

Je dient de concurrentie te volgen. Je voegt twee nieuwe pizza's "clam" en "veggie" toe en je haalt "greek" van de menu

```
public Pizza orderPizza(String type)
{
        Pizza pizza;
        switch (type.toLowerCase())
        {
                case "cheese": pizza = new CheesePizza(); break;
                case "greek": pizza = new GreekPizza(); break;
                case "pepperoni": pizza = new PepperoniPizza();
                                  break;
                case "clam": pizza = new ClamPizza(); break;
                case "veggie": pizza = new VeggiePizza(); break;
                default: pizza = null;
        }
        if (pizza != null){
            pizza.prepare();
            pizza.bake();
            pizza.cut();
            pizza.box();}
        return pizza;
}
```

Deze code is NIET GESLOTEN voor VERANDERING. Als de pizzeria zijn pizza-aanbiedingen verandert, dan moeten we in de code duiken en deze veranderen.

- **De objectcreatie isoleren**

OOAD

```
public Pizza orderPizza(String type)
{
        Pizza pizza;

        if (pizza != null){
          pizza.prepare();
          pizza.bake();
          pizza.cut();
          pizza.box();
         }
        return pizza;
}
```

```
switch (type.toLowerCase())
{
 case "cheese": pizza = new CheesePizza();
                break;
 case "pepperoni":
       pizza = new PepperoniPizza(); break;
 case "clam": pizza = new ClamPizza();
              break;
 case "veggie": pizza = new VeggiePizza();
                break;
 default: pizza = null;
}
```

We pakken de code voor de creatie op en verplaatsen deze naar een ander object dat alleen maar het maken van pizza's als taak zal hebben. Dit object noemen we Factory.

- **Een eenvoudige pizzafabriek**

OOAD

```
public class PizzaFactory {

        public Pizza createPizza(String type) {

           switch (type.toLowerCase())
           {
                case "cheese": return new CheesePizza();
                case "pepperoni":
                        return new PepperoniPizza();
                case "clam": return new ClamPizza();
                case "veggie": return new VeggiePizza();
                default: return null;
           }

        }
}
```

- **De klasse PizzaStore opnieuw bewerken**

OOAD

```
public class PizzaStore {

        private PizzaFactory factory;

        public PizzaStore(PizzaFactory factory)
        {
                this.factory = factory;
        }

        public Pizza orderPizza(String type)
        {
                Pizza pizza;

                pizza = factory.createPizza(type);

                if (pizza != null){ pizza.prepare(); pizza.bake();
                                    pizza.cut(); pizza.box();}
                return pizza;
        }
}
```

PizzaStore krijgt de fabriek die in de constructor wordt overgedragen.

Definitie van de Simple Factory